【研究ノート】

プログラム開発とその生産性

—構造化プログラミングを中心にして—

平　澤　一　郎

目　次

はじめに
Ⅰ　プログラムの効率的開発
　1　プログラム開発技法
　2　プログラム・エラーの根源
Ⅱ　プログラムの効率的手法
　1　プログラム手法
　2　明瞭なプログラムの作成
Ⅲ　わが国におけるプログラム開発
おわりに

はじめに

現代社会において経済発展を促進する原動力の一つに 情報をあげることができる。そのような社会を情報化時代と呼んでいる。

情報化時代という複雑化，高度化した現在の経済の中で，総合的かつ円滑に情報システムの発展を促進するためには，電子計算機のハードウェア（Hardware）およびソフトウェア（Software）の開発に関連する諸問題の効率化および標準化によって生産性を追求することが必要である。特にシステムの多様化，急激な増加，そして作成されたシステムのライフサイクル短縮化の傾向がますます 増大してくる今日では，ソフトウェアの生産性向上がより強く要求されることは明白である。

ソフトウェアの専門家にとっては，信頼性のある，高品質の各種プログラムを，

高い生産性をもって仕上げることは永遠のテーマである。つまり，プログラムの開発とは単にプログラム・コーディングを行うことだけではなく，作成した後のメインテナンスの効率化をも考慮して行われるものであって，従来からも次のようなことが論じられてきた。

「とくに，システム（System）の構造化，文書化，プロジェクト管理およびそれらすべての相互連絡が適切であれば，大規模で複雑なプログラミング・システムの作成，保守を容易にし，機能を拡大することに大いに役立つはずである。またそのことより，システムの寿命も延長することになる(1)。」

このように，各種プログラムは物を作りあげるのとは異なり，人間の頭脳から生み出されるものであって，時間と費用をかければできるというものではない。そこで本稿においては，効率的なプログラムの開発を志向して，プログラム開発時における留意点やプログラム開発の技術的な方法論について述べることにする。さらに現在のわが国におけるプログラムの開発目標はどのようなものであるかについても述べることにする。

1　プログラムの効率的な開発

一般に大型のプログラムの開発は多人数によるグループ作業として行われてきた。この開発方法では完成するまでに時間，労力が増大し，開発費用も尨大なものになってくる。それ故，如何にして効率的なプログラム開発を行えばよいかについて従来からも試行錯誤がくり返されてきた。

現実的にはこのねらいを達成することは容易ではない。それは「この分野の研究は始まってから日が浅く，I．P．T（Improved Programming Technologies：効果的プログラム開発技法）は一つの大きな成果ではあっても，I．P．T自体が日々改良されるべきものとしてとらえられており，まだまだ流動的な面を残している(2)。」とされているからである。

そのため，電子計算機メーカー各社は，プログラム開発方法には尨大な費用を費やし，自社独自のものを発表するように心がけている。特に電子計算機メーカーは，新機種を発表する時に，附随するソフトウェアについては開発時間不足等のため，無

理をして処理に必要なソフトウェアだけをユーザー側に提供しているのが現状である。その後ユーザー等の苦情によりその都度処理（修正）をするような付け焼刃的なことが多い。

しかし，1970年以後徐々にではあるがプログラム開発は向上してきた。それにはソフトウェア開発において次のような研究がなされてきたためである。「プログラム開発作業においてなにが生産性や品質を左右するかを追求して実際の開発作業や設計文書等の分析を行なった。また，プログラムの品質と生産性とは，本来密接に関連し合うものであり，プログラムのエラーの防止，除去と生産性向上は同時にねらえるものである[3]。」これはプログラム開発において生産性と品質は互いに関連し合うものであることを大前提としており，どちらか一方をもおろそかにできないことを示している。

プログラムは作成されてから成熟，終末までのライフサイクルを必らずたどるため，作成時において，第三者が見ても容易に，理解修正できるように仕様書を作成しておかねばならない。それは，要件分析やシステム設計時において特に細かく，綿密に行なわねばならない。

このようにプログラム開発を行なう時点において以上のものを必らず文書化しておかねばならない。しかし，一般にプログラム作成者は文書化を好まないし，うまく他人に理解できるように作成することができない人も多い。

ここで大切なことはプログラム文書化に時間をかければ良いシステムができることである。そのため，作成されたプログラムに問題が発生，または修正を必要とする場合容易に理解，修正することができるのである。このようにプログラムの文書化はプログラミング作業の一部と考えるべきである。その場合に，この文書化を行なうための標準化が重要な問題となる。当然なことであるが，この標準化には，誰が，何を，またどのような形式で文書化するかを明確にしておかねばならない。

1　プログラム開発技法

近年開発するシステム自体が次第に大型化し，複雑化しつつあるため，開発するソフトウェアは，コストの増大，開発期間，要員の問題等と大きな問題が山積みされている。現在ソフトウェア開発費用は，ハードウェア開発費用の2～3倍にもな

っているため，適切なソフトウェアを短期間で開発しなければならない。

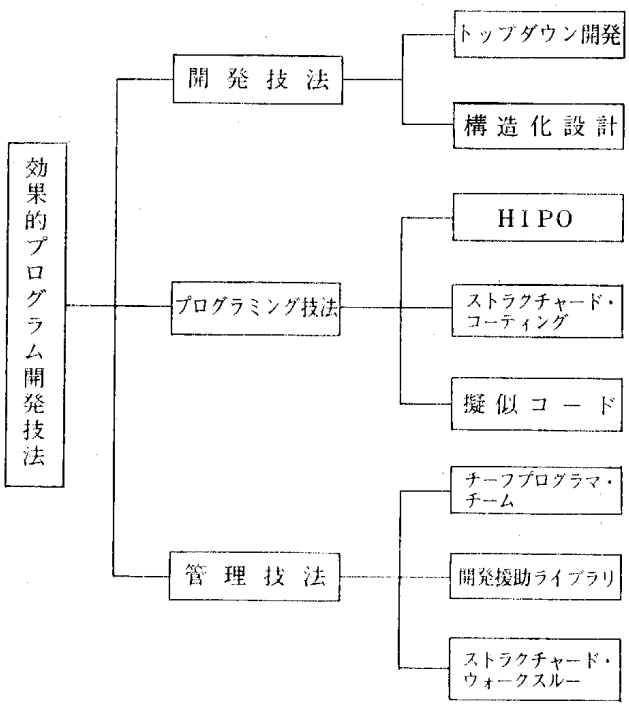
これはシステム開発要求内容の複雑化，および高度化，人件費などの昂騰がその伏線となっている。そのため今や各電子計算機メーカーはハードウェア開発より，ソフトウェア開発の合理化に取りくんでいる。このことは言いかえればハードウェア開発技術より，ソフトウェア開発技術の方がおくれていると言える。また近年になり，システムを構成する要素も年々増大化しつつあり，より複雑で大きなシステムを作成することに対して信頼性を高く，開発および保守コストを低減させることを目的としている。しかも構造化プログラミング (Structured Programming) の思想と特性が発表されてすでに久しいが，これにより特殊技術化されていたシステム開発およびプログラミング技法は今までより科学的に分析，開発できるため，今後はこの技法がプログラミング技術において主流になることは確かである。この構造化プログラミングが一般化され，現在はＣＯＢＯＬプログラムにも適用されるようになった。今までのプログラムは，完成されたものにおいて信頼性および保守性に問題があった。この保守性の問題であるが，単にプログラム修正だけでなく，費用の面を考えねばならない。

現在，わが国において「保守費用が，ソフトウェア費用の⅔以上をも占めているのである[4]。」そのため，いかに効率良くソフトウェアの保守ができるかを考えるべきである（保守の容易さ）。今後は保守をできるだけ容易に，少なくするためのシステムを如何に，作成するかに重点を置く必要がある。このプログラム保守において，効率の良い方法は，構造化されたプログラムである。プログラム保守というものは，作成されたシステムまたはプログラムのライフ・サイクルをなすもので，大きな費用を併うのが普通である。また，プログラムの保守を行なうスタッフとしては，プログラミングの経験の少ない人に任かせるのが普通である。そのために，効率の良いプログラム開発技法として次のような体系が示されている。

この効果的プログラム開発技法の体系は，プログラム作成時における，品質と生産性の向上，保守の容易性等を考え合せた上で発表されたものである。

特に図１で示したような開発技法のなかで最も実施し易い方法は，ＨＩＰＯ (Hierarchy Plus Inpur Process Output)，構造化設計 (Structured Design)，およ

図1　効果的プログラム開発技法の体系



藤田栄保他著「効果的プログラム開発技法」導入の評価
Fujitsu 1978, Vol. 29, No. 6, p. 235

びストラクチャード・コーディング (Structured Coding) である。

ＨＩＰＯは，ＩＢＭ社が開発した文書化する際に用いる図式表示技法である。また大規模なオペレーティング・システム開発を目標に開発され，システム設計を行なう場合要件分析にも有効である。この技法は階層的なものと，プロセス的なものから形成される図的表記法とに分かれる。この表記法は系統立って作成されるためプログラムの流れ，および内容をチェックが容易となる。

今までのプログラミング時において作成されていたフローチャート (Flow Chart) は，大規模かつ複雑化されたシステムに利用するには，保守等の容易さを考えると良い方法とは言えない。

プログラムは可能な限り細分化（モジュール化）し，互いのモジュールの関連づけを確実にしなければならないからである。そのため，プログラム仕様書段階において処理別に細分化することが重要である。その理由として，プログラムは可能な限り第三者が理解できるよう分割をする。また，分割されたモジュールが互いに関

連を持たねばならない。この分割されたものは一つの構造を成していなければならず，一つの構造は修正等の保守を容易にするため 50～100 ステップ程度に押さえておく。

ストラクチャード・コーディングとはプログラム作成方法のテクニックの一つであり，一般にはストラクチャード・コーディングとか Go To Less プログラミングと言われている。しかし Go To 文を使用したための弊害は一切ない。

一つのモジュール中において Go To 文は正しいプログラム仕様書が作成されていれば不必要になることは明白である。そのため，Go To 文を使用せねばならない場合が発生した時大部分はプログラム・プランニングの失配によるものである。

次に各プログラム技法の評価について藤田栄保氏の調査から引用すると表1のようになる。これら各技法の導入はそれぞれ各自（社）独自の判断と状況に応じて最適なものを取り入れればよい。本表では非常に良く使われている技法か，あまり使われていないかに大別されている。

表1 各 技 法 の 評 価

	技法	導入目的	評価		
			長所	短所	短所の対処案
非常によく使われている技法↑	HIPO	仕様書の標準化 仕様書のメンテナンスの容易性	コーディングがやりやすい仕様書が見やすい 論理が追いやすい	記述レベルを統一して設定することが難しい	規約の設定と教育の徹底
	構造化設計（SD）	プログラム構造の最適化 プログラムを均一化し信頼性の向上	機能単位にモジュール化を図るため，プログラム構造が明確になる レビューがしやすい テストケースが洗いやすい	機能分解の手順が難しい 慣れるまで時間がかかる	手順の標準化と教育の徹底
	ストラクチャード・コーディング（SC）	ソースリストが最終のドキュメントという考え方から，できる限り標準化を図ることにより属人性の廃止を狙う	第三者にも非常にソースリストが見やすくなる 機能とロジックが明確に区分できる ソースリストのより一層のドキュメント化が図れる	現状の言語レベルでは徹底したSC化ができない部分もある	コンパイラ言語のSC化への改善

ストラクチャード・ウォークスルー（SWT）	仕様ミスの防止 要員全体の知識レベルの統一	標準化，機能，論理，インタフェースの誤りの早期発見ができる より多くの観点（複数メンバ）でのチェックができる	ウォークスルーの工数確保が難しい 時間を区切って行うことは難しい	ウォークスルーの運用の工夫
トップダウン開発（TDD）	単体テストを行いながら結合テストができるため，開発の効率化を狙う	デバグ効率が向上する インタフェースミスが早期に発見できる	上位モジュールの遅れが全体に影響を与える スタブ 注）作成のめんどうさが残る 進捗管理が難しい	方法論の確立
チーフプログラマ・チーフ（CPT）	各サブシステム単位に作業分担を明確にする チーム全体での問題対処と機動力の充実	大規模プロジェクトの分業化に最適である 全体の管理能力が向上する 作業分担が明確になる	チーフプログラマの人材が不足している チーフの管理能力によりチームが左右される チームの機能が十分生かされるまで時間がかかる	チーフプログラマ・チームを受け入れる土壌の整備とチーフプログラマの育成
開発援助ライブラリ（DSL）	設計者，プログラマの事務作業からの開放 管理面での一元化 開発プログラムの削減	外部，内部ライブラリの一元化により開発負荷を低減できる ライブラリの最新状態が常に明確である 設計者，プログラマの事務作業を軽減できる	管理手順，ツール作成の工数確保が難しい ライブラリアンの確保が難しい	管理手順，ツールなどの事前準備とライブラリアンの教育
擬似コード（PC）	コーディングミスの防止 コーディングのスピードアップ	仕様がより明確となる 仕様作成時，作成者自身の頭の整理となる	擬似コードの位置づけと規約設定が難しい ドキュメント量が多くなる	記述言語からオブジェクトコードを生成するようなツールが必要

←あまり使われていない技法

藤田栄保他著「効果的プログラム開発技法」導入の評価
Fujitsu 1978, Vol. 29, No. 6, P. 241

2　プログラム・エラーの根源

プログラムを開発するに当たり，発生するエラーにはどのような原因があるか，またプログラム・エラーとはどのようなものであるかを述べる。エラーが発生する場合の主な原因は，プログラム開発依頼元より提出された仕様書通りに作成しなか

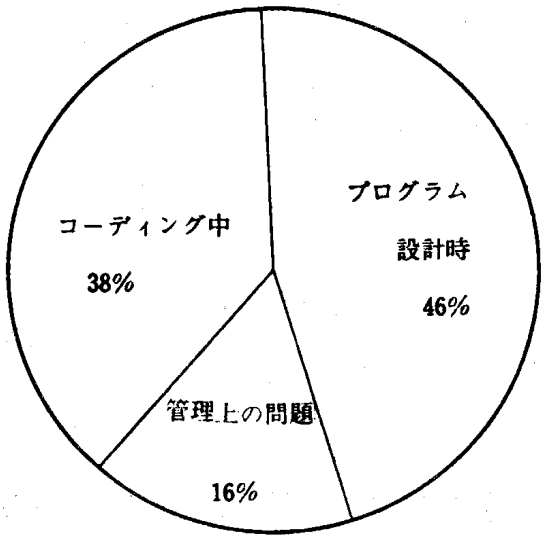
った場合，または仕様書作成者が特別な処理がある場合，その処理を見落して仕様書を作成した場合作成されたプログラムは定常の処理は行なうが，特別な処理（例外処理）は行なわない。これもプログラム・エラーである。

即ち，仕様書を作成するに当たり，当然常に正確に作成することを前提としているが，実行されない例が多いのである。しかし，現状において，この前提が正しく書かれていること自体稀であり，プログラムを書きながら修正を行なうことが多いのである。

またシステム設計時においてコンピュータのハードウェアの内容を考えず，限界を越えるような仕様書を作成した場合，プログラムは行動するが実際にはそのプログラムは業務処理には役立たない。これもプログラム・エラーの一つである。

次にプログラム・エラー発生の根源別構成を，1975年ＤＯＳリリースの例でＩＢＭ社が調査した結果を示すと次のようなものである。（図２）

図２　プログラム・エラー根源構成比



(Access March 1980, p. 13)

エラーの原因は開発時点で作業項目をチェックすることによって捉えることができる。その作業項目には次のようなものがある。

(1)　システム設計時の誤り。

(2)　仕様書等作成時における文書化の誤り。（説明不足）

(3) コーディングおよび修正時の誤り。

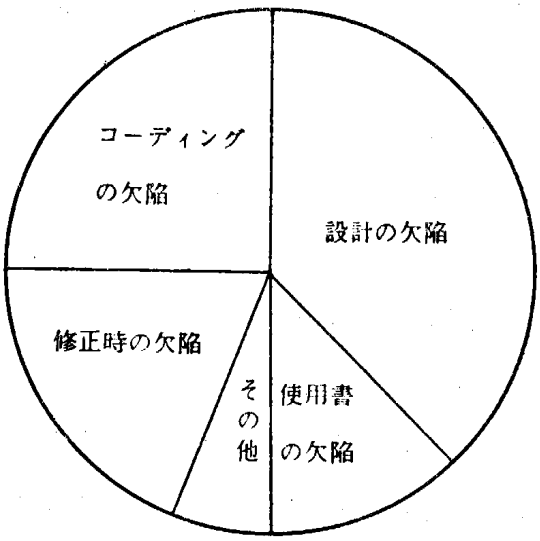
(4) その他

特にシステム設計の誤りはプログラム作成時においては根本的な誤りとなり，致命的なものである。この低い水準の設計はプログラム・コーディング時にエラーを多発させることにもつながるものである。

「このエラーが多くの異なる部分でコーディングをしている人々に伝播すると一つの設計の誤りがシステム内の多くの誤りとなる[5]。」

ここでプログラムエラーがどの部署（時点）で多く発生したかを，ＩＢＭ・ＤＰプロダクト・グループによって調査された結果を図示する。（図３）

図３ 欠 陥 の 種 類



(Access April 1980, p, 5)

これを見るとプログラムを作成する時点において50%近くが，システムおよびプログラム設計時に発生している。またプログラム・コーディング中におけるエラー発生は全体の25%である。ここで最も困るエラーは誤りを直そうとして発生するエラーが25%もあることである。

従ってプログラムを作成する時点において致命的なエラーとしてプログラム設計時のミスが多い。そのため，細心の注意を払い完全なる仕様書を作成しなければならない。

今後プログラミング時においては仕様書のあいまいさ，矛盾のある仕様書，または重複，その他の業務を考え合せて融通性のあるものを作成せねばならない。

Ⅱ　プログラムの効率的な手法

プログラムを作成することは文章を書くことと同一と考えるべきである。

常に他の読み手を意識し，相手の立場に立って考え，決して一人よがりの記述をすべきではない。また作成したプログラムの読み手（処理も含む）は計算機と人間であることを忘れてはならない。

計算機のためには，決められた文法に従がい正確に書かねばならない。これには点（ピリオド）1ケの誤りも許されないほど厳密なものである。人間のためには読み易く，特に理解し易いプログラムを作成する必要がある。

ここでは，ＣＯＢＯＬ言語について特に説明をする。そのため，プログラム作成にあたってはその特性を十分に生かしてプログラムを作成すべきである。

単にプログラムと言っても前述したように単に書くだけでなく，ハードウェアと同様，今後はソフトウェア（ここではユーザープログラムに限定する）も部品化（細分化）することにより拡張性，汎用性，保守性を考えて作成すべきである。

ここでプログラムの細分化とは，ある一定の処理機能を持つものを指し，プログラム作成時において，その要求される処理機能はどの要素から構成され，どのように分割されているかが問題である。分割されたプログラムにおいて，特にメイン・セクション (Main Section) とサブ・セクション (Sub Section) に分割し，このうちでプログラム全体を制御する部分をメイン・セクション，その他のセクションをサブ・セクションとする。サブ・セクションはプログラム作成者によって分割され，サブ・ルーチン (Sub Routine) の役割をするものである。これを図で示すと図4，図5，のようになる。

このように分割しても，一つのセクションが大きすぎた場合解読が大変なため，一つのセクションは平均して 50～100 ステップ程度を標準とするのがよい。これ以上大きくなる場合は，その一部を別のセクションとして独立させ，ステップ数を調整することを考えればよい。

図4　プログラムのセクション関連図

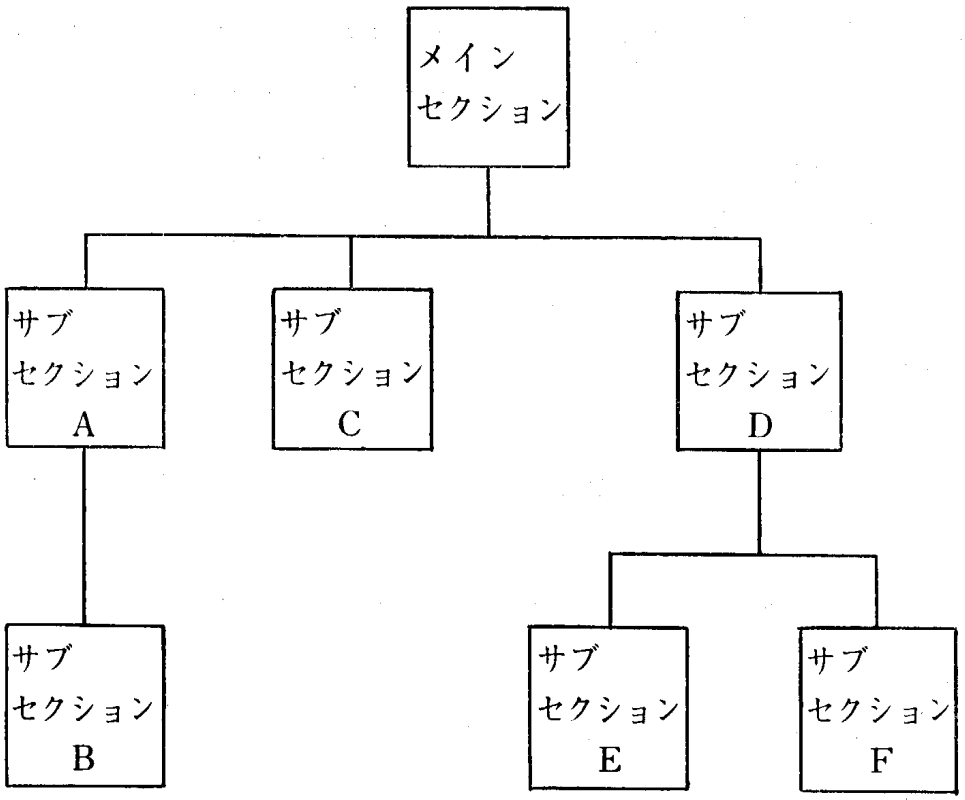
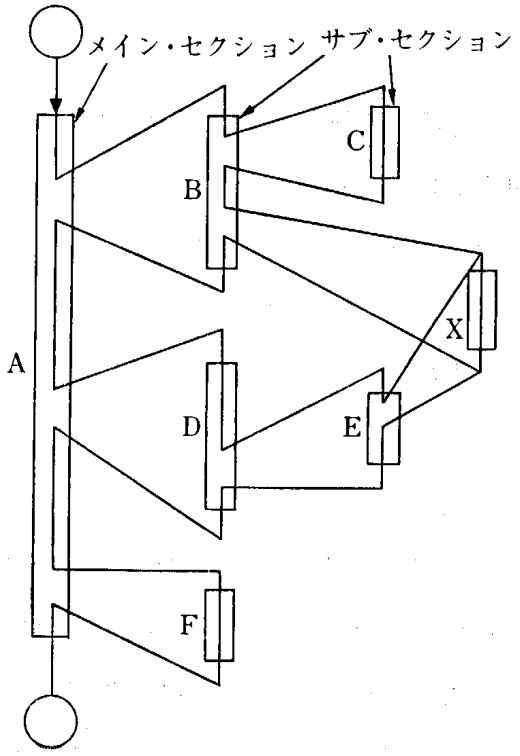


図5　プログラムの制御の流れ



1　プログラム手法（構造化プログラム）

今までのプログラミング方法は，与えられた条件（仕様書）に従い無条件にプログラム・コーディングを行ない，使用する命令には何ら制限はなかった。

しかし，構造化プログラミングという新しい方法が発表された時点において，ソフトウェアを担当する人々に大きな衝撃を与えたのである。

これは，プログラミング技術，手法を大いに前進させるものとして評価できるものである。また企業内（ユーザー側）において構造化プログラミングを利用する場合，その方法論は新しいものであっても，システム分析を行なう場合，以前と変らない方法をとっている。それ故構造的方法で分析を行なったと思われる設計において，重要な点を欠いたり，誤ったりするため以前とはあまり変っていないのが現状である。またこの構造的分析方法を各企業内において利用する場合，本当に有効であるかどうかの評価を与えることは現在においては早計である。

構造化プログラミングとは一般に分岐命令（Go To）文を省略したプログラムであると誤解されている。しかし，必要に応じて使用することは何ら差しつかえない。また，このコーディング基準は構造化設計によって作られた（モジュール化）プログラムを実際にＣＯＢＯＬ言語によってコーディングされる際用いられるものである。

「プログラムとその実行の流れに密接な関係がある分岐（Go To）命令を不注意に使用すると，この関連性が妨げられる。

それ故，構造化プログラミングを“Go To なしの（Go To-Less）”プログラムと言うこともある。しかし，Go To 文を使用せずきちんと行のあげさげをつけたプログラムでも，まったく理解しにくいものもあれば，Go To 文を含んでいても完全に理解できるプログラムもある。従って Go To 文の有無はプログラムの良さの適切な尺度とはいえない[6]。」

構造化プログラムの方法は現在も試行錯誤の時であり，完成されたものでないため，構造化プログラムを利用してプログラミングを行なう場合，この変化に対応できるようコーディングをする必要がある。この構造化プログラムを作成する場合，一般に行なわれている方法は，今まで作成していた構造化されていないプログラム

を作成した後に，そのプログラムを構造化プログラムに修正するという二重手間を行なっている場合が多い。そのため，構造化プログラムを作成するには，まず基本的な構文を理解することが大切である。また，ここで重要なことは，構造化設計および構造化プログラム作成の主な目的は，拡張性，信頼性，保守性の高いプログラムを作成することにあり，これ等の問題を損ねない限りプランニングを行なうときに，多少作成時に変更をしても良い。このプランニングにおける問題は基準を設定し，それを徹底させる方が特に重要である。

構造化プログラムは前述したように，未だに進歩の段階であるが，基本型は変らないため，ここに構造化プログラム作成時の基本構文の定理をあげておく。

図6　構造化プログラミングの基本構造

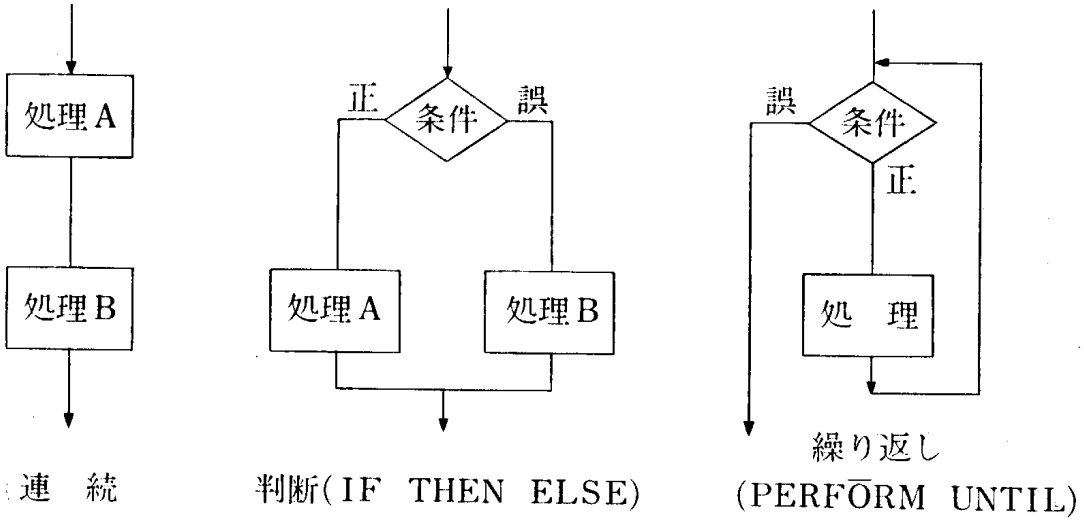


図6に示した判断命令のＥＬＳＥについて説明する。前述した通りＣＯＢＯＬプログラム中において Go To 文を完全に無くしてしまうと，何重にもＩＦ　ＴＨＥＮ　ＥＬＳＥ　文を使用しなければならなくなり，プログラムの基本的構成は一般に理解しにくくなる場合が多い。これではプログラム中において最も重要である条件チェックを行なうには非常に困難になる。そのため「ネスト（ＮＥＳＴ）したＩＦ構文とＥＬＥＳ節とを避けることが望ましい。ＥＬＳＥ節はＩＦ　ＴＨＥＮで条件の否定を用いる場合と等価なことから，ＥＬＳＥ節は通常は不必要なことは明ら

かである[7]。」といわれる。

このように今後のＣＯＢＯＬプログラムは Go To 文およびＥＬＳＥ節を省略し，ＥＪＥＣＴ命令やＳＫＩＰ命令を用いて第三者が理解し易いプログラムを作成することである。特に，プログラム作成時において，どのようなプログラムであろうと，エラーを発生させるのは計算機自体ではなく，人間がコーディングしたプログラム・ミスによるものであるため，作成過程において，より明瞭な分析をする必要がある。ここに書いた構造化プログラミングの方法はプログラム作成者にとって，あくまでも絶対的な方法でないことを忠告すると共に，今迄作成していた普通の文体についての大切さを再認識すべきであるという説もあることを忘れてはならない。

構造化プログラミングに関する種々の技法の一覧を示すと次の通りである。

表２　ストラクチャード・プログラミング技法

ＳＰ　手法の名称	開　発　者
Structured Programming	Dijkstra 氏
HIPO	IBM 社
JACKSON	Jackson氏 (INFOTECH 社)
SSDM	Boeing 社
DECA	〃
SACT	Soft Tech 社
SARA	G. Estrin 氏 (UCLA)
YOUNDON/CONSTAN-TINE の構造設計法	Yourdon/Constantine 両氏
CHAPIN CHART の方法	CHapin 氏
PARNAS の方法 SOFTWARE ELUEPRINT	Yoahan Chu 氏
ARDI	Philips 社
Young & Kent の方法	Young/Kent 両氏
TAG	IBM 社
ADS	NCR 社
Bertini & Tallineau の方法	Bertini Tallineau 両氏
Warnier 法	Honeywell-Bull 社

（社団法人　日本能率協会　ＥＤＰリサーチ・レポート，1980.2.1）

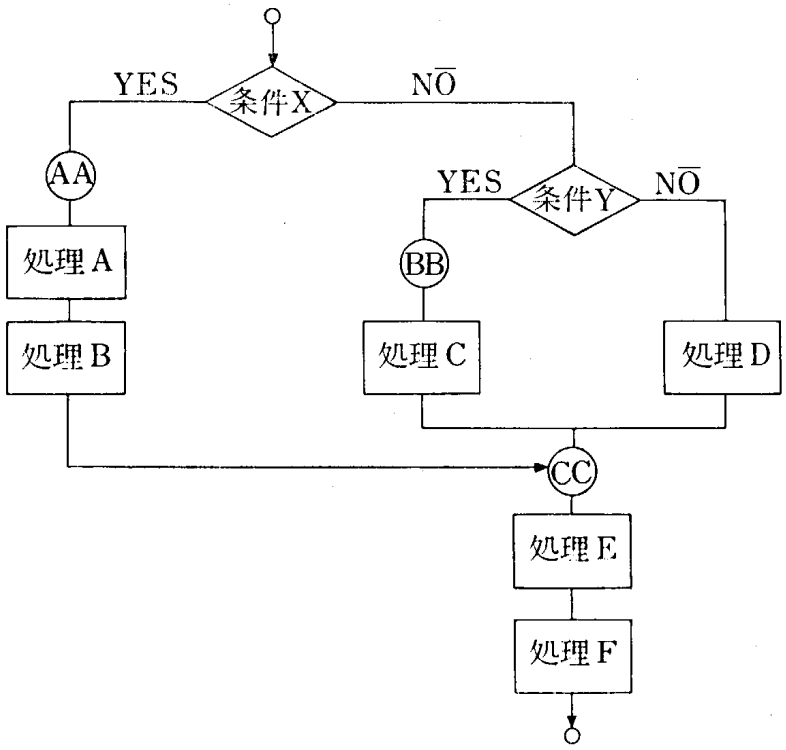
２　明瞭なプログラムの作成

プログラムを作成する場合，目標は使用する機械のためではなく，いかに人間が

理解できるようなプログラムを作成するかにある。それには，仕様書の正確さ，プログラムの明瞭さ，理解し易さ，および保守のし易さにある。

プログラム作成時において，仕様書の内容をどのように分解して作成するかは，そのプログラム担当者に一任されるが，分解方法によって，そのプログラムの理解し易さは大きく左右される。またそのプログラムの保守性にも大きく影響する。

明瞭なプログラム作成の例を示すと次の通りである。



〔一般的なコーディング例〕

```
iF 条件X（ＹＥＳの時）Go To ＡＡ.
iF 条件Y（ＹＥＳの時）Go To ＢＢ.
  処 理 D .
  Gō To ＣＣ.
ＡＡ.
  処 理 A .
  処 理 B .
  Go To ＣＣ.
```

```
BB.
        処 理 C.
CC.
        処 理 E.
        処 理 F.
```

〔構造化プログラムの例〕

```
    iF 条件 X
        THEN
            処 理 A
            処 理 B
        ELSE
            iF 条件 Y
                THEN
                    処 理 C
                    処 理 D.
    処 理 E.
    処 理 F.
```

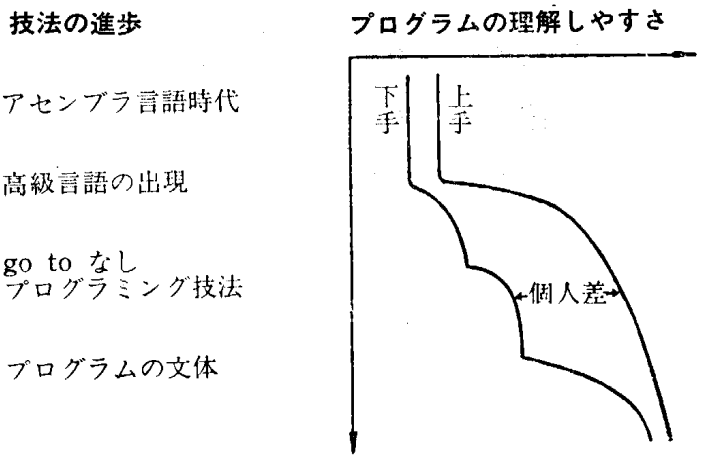
このようにCOBOLプログラム作成時において，モジュール化することは，プログラムの複雑化をさけるための最も基本的な方法である。そのほかに一般的なコーディング例で示したように，1個または2個のモジュールを処理する毎に Go To 文で他のモジュールへ飛ばして処理を行なう。このように飛び先を変えて，コーディングする場合，後からそのプログラムを保守する場合はミスが発生し易い。このため構造化プログラムの例で示した方法によると，条件に当てはまったモジュールの処理を考えれば良いため，プログラム自体読み易く，保守もし易いのは明白である。

このようにプログラム作成時において，モジュール化することはプログラムの複

雑化をさけるための最も基本的な方法である。これら，モジュール化された構文は互いに密接な関連をもたせることも必要であるが，他のモジュール化された構文とはできる限り独立をさせなければならない。

このような考え方が発展したために，前述した構造化プログラミングが注目をあびるようになった。また，プログラム作成者は自分の能力により種々の工夫および創作ができるようになった。プログラム作成者の能力により，プログラムの良し，悪しがはっきりしてきた。「プログラムの保守性の保証・向上が系統的にできるようになり，個人差は上手（プログラムの性能および技法の良い方）に縮まりつつある[8]。」（図7参照）

図7　プログラムの理解しやすさの進歩



（君島浩，河田汎著「ソフトウェア開発における品質向上の技法」Fujitsu Vol. 29, 1978, p. 220）

このように今後のプログラムは構造化プログラミング技法により，プログラム作成者の能力によって或る程度上手，下手のギャップはうめられるにしても，画一化された，理解し易く，保守性のある方向に進み，誤りも発見され易くなる。

しかし，ソフトウェア技術者（狭義にはプログラム作成者）によってプログラムの品質は左右されるという個人差が生ずるのは当然である。そこで，より良いプログラム開発方法および，より明瞭な，保守性のあるプログラム作成方法の必要性とその開発の重要性がある。

Ⅲ　わが国におけるプログラム開発

多種多様の情報化ニーズを満たすため，ソフトウェア開発は益々複雑となり，近年とみに情報処理経費に占めるソフトウェア開発費用の比率が増大しており，それに何らかの対処をしなければならない時期にきている。

今までのソフトウェア開発は前近代的であり，手工業的かつ注文生産的な，他に利用できないようなこれっきりの生産が続いてきた。それを解決するためにはソフトウェア開発時点の省力化，合理化，および生産性の向上をはかるとともに，有能な人材の養成が主眼となる。

特にわが国におけるソフトウェア開発は，方法論のみならず実務的にもアメリカにおけるソフトウェア開発のレベルより数年おくれている。そこで通産省がはじめて，ソフトウェア開発目標を「プログラムの開発に係る電子計算機利用高度化計画」として発表したのが昭和47年1月である。

それの第一次完成目標を昭和50年度中とし，それによると開発プログラムの種類は，次の通りである。

1　制御プログラム

2　通信制御プログラム

3　言語プロセッサ

4　ユーティリティ・プログラム

5　アプリケーション・プログラム

上記のプログラムについて試作，実用化，性能向上，または汎用化を行なうことが主眼である。

続いて昭和51年3月26日には通産省，郵政省，告示第1号により，今までより明確にわが国のソフトウェア開発の目標，計画が発表された。

それには，今後5年間の開発目標が指示されており，データ量の増大化に併うための処理方法の一つとして，データ・ベースの定義が明確化されており，いよいよデータの大量化，複雑化に対応できるような計画が含まれている。これは，今後の実用化への大きな指標であると共に，わが国でデータ・ベースへの取組み方を明示したこ

とは特記すべきことであった。（特に開発完成目標を昭和55年度中と指定してある）。

またデータ・ベースの利用方法として異機種間との関連を持たせること，医療関係にも利用できるような汎用性を持たせること等と細かく具体的に指示してある。

この第二次計画におけるプログラム開発の種類は，

1　社会開発用プログラム

2　特殊情報処理・検索用プログラム

3　経営計画・管理用プログラム

4　流通・サービス用プログラム

5　生産用プログラム

6　科学技術計算用プログラム

である。

この発表された計画には，言語関係について一切ふれていない。これは通信制御言語，ユーティリティ，アプリケーション等と処理が全く異なるため，開発者（特にメーカー）の判断にまかせることを示している。

今後はアメリカとの技術格差をなくし，情報処理量の増大化に対処し，プログラム（広義においてソフトウェア）の評価方法を確立すると共に，より明確な標準化を図るべきである。また，データの保護対策の必要性，および環境の変化に直ちに対応できるような，ソフトウェアを開発することを，あわせて考えねばならない。

おわりに

電子計算機の技術発展は年々その速度を増し，その結果として性能が向上すると共にコストは低下している。当然のこととして電子計算機の設置台数は近年飛躍的に伸びている。

それに比較してソフトウェアの発展は遅々としたものであり，プログラミングの標準化および統一化のおくれも大きな原因となっている。

これらの状況を考えて見ると，より大きなシステムを作る場合における信頼性を高め，開発コストを低減させるための研究が盛んであるが，現状ではこれらの問題

に対し，確固たる結論が出ていない。ところがシステム開発期間，工数は増大化する一方であり，情報処理のコストも年々増大化の傾向にある。

近年になって構造化プログラミングという方法を各電子計算機メーカーおよび，ユーザーが利用するようになった。今後はこの方法が，ソフトウェア開発時点で主力となると思われるため，統一された方法を考え出さねばならない。そのことにより，プログラム・ミスの大幅な減少が期待され，保守も楽になることは明白である。特にソフトウェア開発において，今までのような複雑な方式より脱皮することができ，生産性はより向上することになるであろう。

註(1) Glenford J. Myers 著，Reliable Software Through Composite Design
久保　未沙
国友　義久　訳
「高信頼性ソフトアェア—複合設計」（ソフトウェア設計の現状）p. 1
近代科学社発行（昭和54年12月）

(2) Access March '80, p. 12,「エラーはなぜ起るか」日本アイビーエム㈱編
Vol. No. 3, 通巻113号

(3)「エラーはなぜ起るか」前掲書　p. 12～p. 13

(4) 藤田栄保他著「効果的プログラム開発技法」導入の評価
Fulitsu 1978, Vol. 29, No. 6, p. 235

(5) Access April '80, p. 5～p. 6，ＩＢＭ　ＤＰプロダクト・グループ，プログラミング・プロセス担当部長，アルフレッド M. ピエトラサンタ著「影響が大きい設計時の誤り」日本アイビーエム㈱編，Vol. 6, No. 4，通巻118号

(6) Glenford J. Myers 著，Software Reliability-Principles and Practices
有沢　誠訳「ソフトウェアの信頼性」（ソフトウェア・エンジニアリング概説）p. 149～p. 150，近代科学社発行（昭和53年２月）

(7)「ソフトウェアの信頼性」，p. 157，近代科学社発行，前掲書

(8) 君島　浩　著「ソフトウェア開発における品質向上の技法」p. 220,
河田　汎
Fujitsu 1978, Vol. 29, No. 6